Projeto de Aplicação de REDES – Automação com ESP32 e Java

Ugo Rangel Gemesio

Sistema de monitoramento e automação baseado em ESP32 para coleta de dados e tomada de decisão, com Java + Spring Boot como backend e uma interface simples via Thymeleaf.

Este projeto demonstra integração de dispositivos IoT com serviços backend, cálculos estatísticos, comunicação em tempo real e armazenamento persistente.

Objetivo

Criar uma solução para gerenciar de modo controlável e escalável ESP32 via backend JAVA

- O ESP32 coleta dados de sensores, executa decisões programáveis e mantém uma interface de usuário local (display, WebServer, etc.).
- O Backend em Java (Spring Boot) recebe dados dos ESP32, realiza cálculos como métricas de rede (RTT, jitter, throughput), persiste dados e envia atualizações ao frontend.
- O Frontend em Thymeleaf exibe valores em tempo real e permite ilustrar o conceito de integração IoT + backend.

Objetivo

A aplicação busca demonstrar aspectos de Redes de Computadores, como:

- comunicação TCP/WebSocket,
- ACKs,
- latência,
- estatísticas de transferência,
- consistência de tempo,
- e persistência de dados.

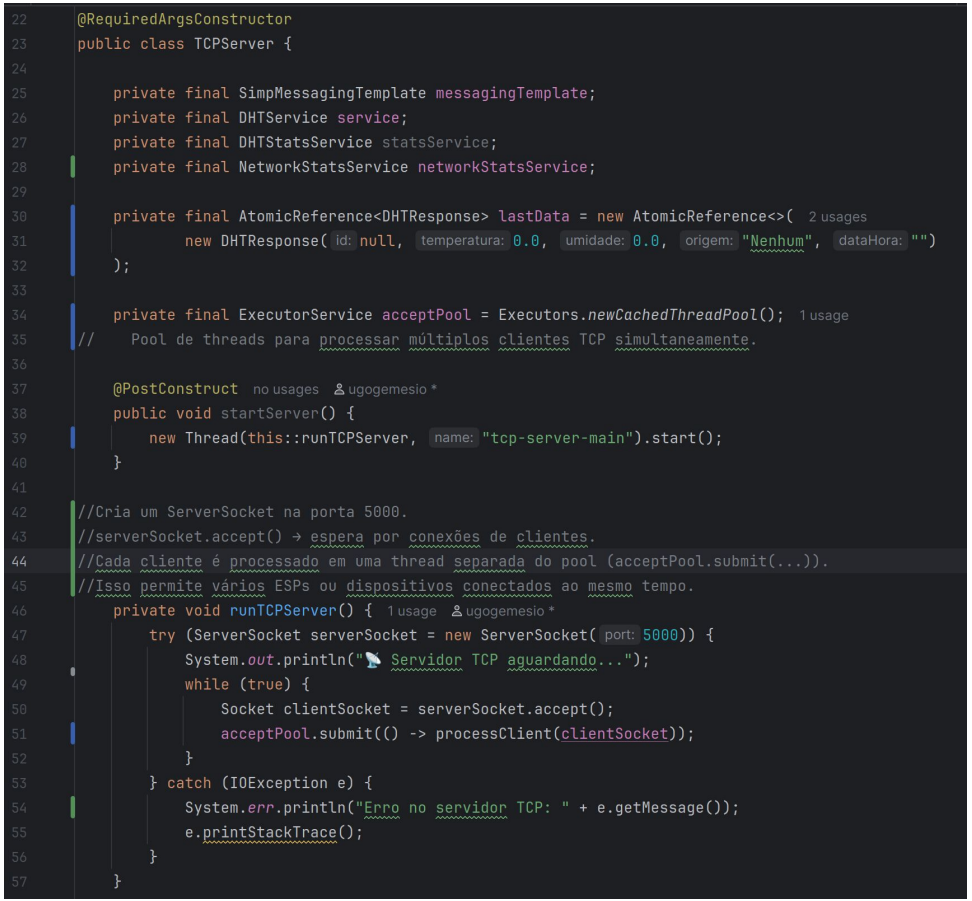
```
// ---- 1. CALCULA RTT EM MICROSSEGUNDOS
unsigned long startPing = micros();
tcpClient.println("PING");

unsigned long timeout = micros();
bool ackReceived = false;

while ((micros() - timeout) < 1000000UL) { // timeout 1s em micros
    if (tcpClient.available()) {
        String ack = tcpClient.readStringUntil('\n');
        lastRTT = micros() - startPing; // RTT em micros
        ackReceived = true;

        if (lastRTT > 0) {
            rttHistory[rttIndex] = lastRTT;
            rttIndex = (rttIndex + 1) % MAX_RTT_HISTORY;
            if (rttCount < MAX_RTT_HISTORY) rttCount++;
        }
        break;
    }
}
```

Funcionalidades - ESP32

- Envio periódico de dados (sensores, métricas, status, etc.) via TCP ou WebSocket. Realiza tomadas de decisão locais (ex.: acionar atuadores).
- Cálculo local de timestamps para RTT.

Funcionalidades - Backend

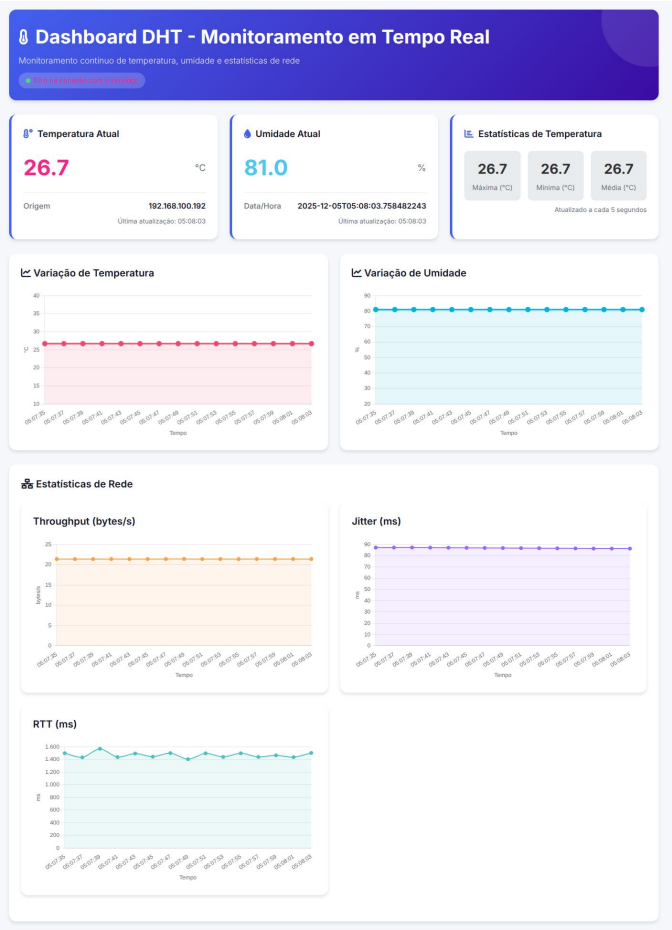
- Recebe dados dos ESP32 via socket/TCP.
- Processa métricas:
- Envia dados ao frontend em tempo real (STOMP WebSocket).
- Persistência em banco (in-memory, PostgreSQL ou outro).
- API REST.



```java
@RequiredArgsConstructor
public class TCPServer {

    private final SimpMessagingTemplate messagingTemplate;
    private final DHTService service;
    private final DHTStatsService statsService;
    private final NetworkStatsService networkStatsService;

    private final AtomicReference<DHTResponse> lastData = new AtomicReference<>( 2 usages
            new DHTResponse( id: null, temperatura: 0.0, umidade: 0.0, origem: "Nenhum", dataHora: "")
    );

    private final ExecutorService acceptPool = Executors.newCachedThreadPool(); 1 usage
//    Pool de threads para processar múltiplos clientes TCP simultaneamente.

    @PostConstruct no usages  ugogemesio *
    public void startServer() {
        new Thread(this::runTCPServer, name: "tcp-server-main").start();
    }

//Cria um ServerSocket na porta 5000.
//serverSocket.accept() → espera por conexões de clientes.
//Cada cliente é processado em uma thread separada do pool (acceptPool.submit(...)).
//Isso permite vários ESPs ou dispositivos conectados ao mesmo tempo.
    private void runTCPServer() {  1 usage  ugogemesio *
        try (ServerSocket serverSocket = new ServerSocket( port: 5000)) {
            System.out.println("📡 Servidor TCP aguardando...");
            while (true) {
                Socket clientSocket = serverSocket.accept();
                acceptPool.submit(() -> processClient(clientSocket));
            }
        } catch (IOException e) {
            System.err.println("Erro no servidor TCP: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

Frontend Thymeleaf

- Dashboard com gráficos em tempo real.
- Controlar esp32 (a fazer).
- Visualização de pacotes recebidos.



Arquitetura do sistema

ESP32 -> Backend Spring Boot -> Frontend Thymeleaf

Sensores
Atores

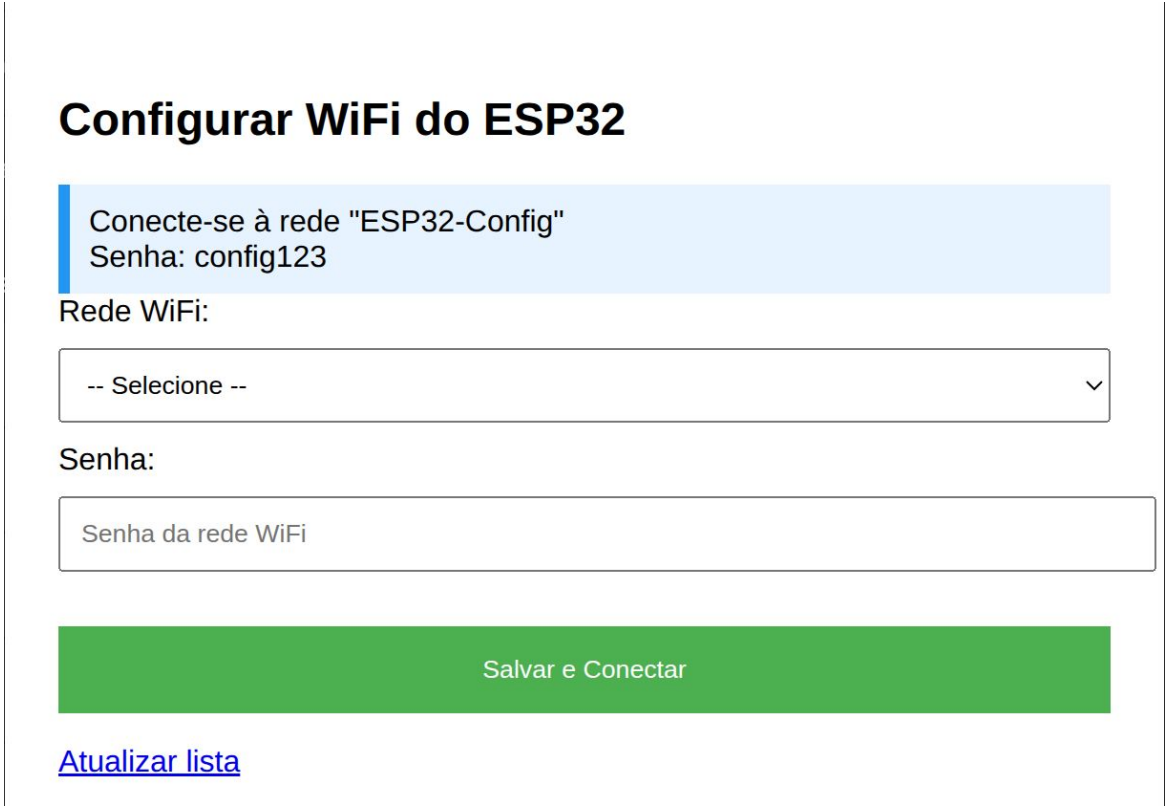
Processamento
Persistência

Visualização
em tempo real

Fluxo de Uso

0. Usuário registra o ESP32 no wifi onde o backend está rodando
1. ESP32 inicializa sensores e conexão.
2. Envia dados periódicos ao backend (timestamp, valores, ACK, sequência, etc.).
3. Backend processa dados, calcula métricas e armazena.
4. Backend envia atualizações ao frontend via WebSocket
5. Usuário visualiza métricas em tempo real.
6. Usuário controla ESP32 via frontend (ainda não implementado).

Usuário registra o ESP32 no wifi onde o backend está rodando



ESP32 inicializa sensores e conexão e Envia dados periódicos ao backend

```
=== ESP32 IoT Gateway ===
Inicializando...
Sensor DHT11 inicializado
Sistema de configuração pronto
Conectando a WiFi salva: dlink-EA1C
...
Conectado! IP: 192.168.100.192

=== MODO NORMAL ATIVADO ===
Servidor HTTP iniciado na porta 80
Acesse: http://192.168.100.192
Conectado a: dlink-EA1C
Enviando dados para TCP: 192.168.100.149:5000

Sistema pronto!
Conectando ao servidor TCP... conectado!
Enviando dados -> 26.7,81.0,192.168.100.192,4129954,34597,0
Enviando dados -> 26.7,81.0,192.168.100.192,6157976,1634,32963
Enviando dados -> 26.7,81.0,192.168.100.192,8186028,1688,16508
Enviando dados -> 26.7,81.0,192.168.100.192,10213856,1512,11064
```

Backend processa dados, calcula métricas e armazena.

```
   RTT atual (ESP): 1471.0 µs
📥 Recebido de /192.168.100.192:61381: 27.1,79.0,192.168.100.192,133919871,1437,60
Hibernate: insert into dhtentidade (data_hora,origem,temperatura,umidade,id) values (?,?,?,?,default)
Hibernate: insert into network_stats (jitter,rtt,throughput,id) values (?,?,?,default)
📊 Network Stats:
  - Throughput: 21.316075197917996 bytes/seg
  - Jitter (média do servidor): 231.337 µs
  - RTT atual (ESP): 1437.0 µs
```

Backend envia atualizações ao frontend via WebSocket

```java
public void configureMessageBroker(MessageBrokerRegistry registry) {
    registry.enableSimpleBroker( ...destinationPrefixes: "/topic");
    registry.setApplicationDestinationPrefixes("/app");
}
```

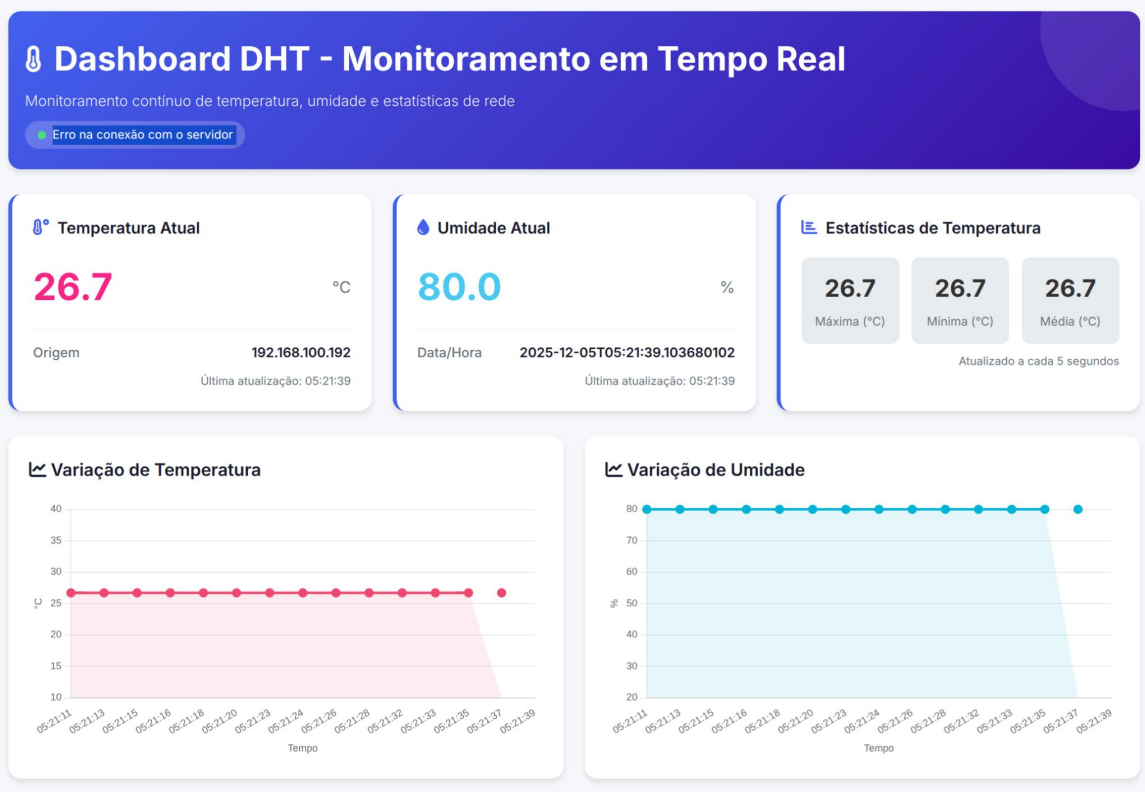
```java
// Prefixo setado em WebConfig /topic envio
messagingTemplate.convertAndSend( destination: "/topic/dht", newLast);
```

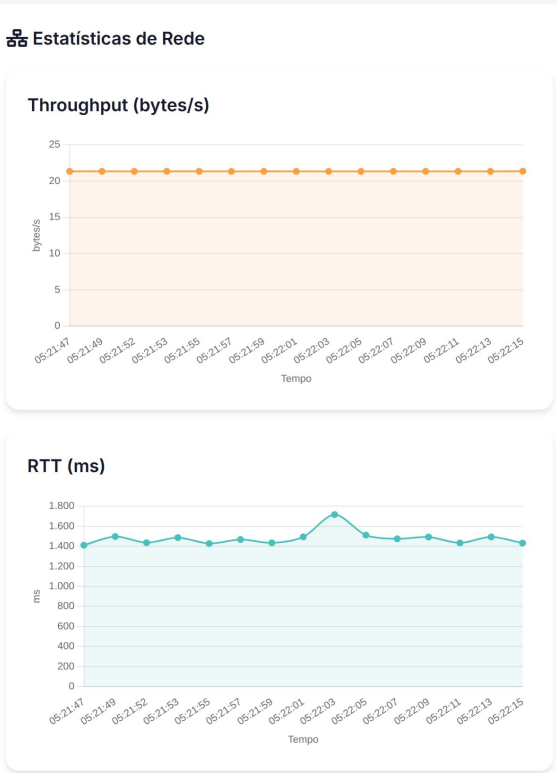
Backend envia atualizações ao frontend via WebSocket

```java
public void configureMessageBroker(MessageBrokerRegistry registry) {
    registry.enableSimpleBroker( ...destinationPrefixes: "/topic");
    registry.setApplicationDestinationPrefixes("/app");
}
```

```java
// Prefixo setado em WebConfig /topic envio
messagingTemplate.convertAndSend( destination: "/topic/dht", newLast);
```

5. Usuário visualiza métricas em tempo real.





Execução do Projeto

Pré-requisitos

- Java 25
- PostgreSQL se quiser persistência
- PostgreSQL se quiser persistência,
- Gradle,
- ESP32
  - DHT11 - sensor de temperatura e umidade utilizado como exemplo.
  - qualquer outro módulo que se queira controlar e interagir.

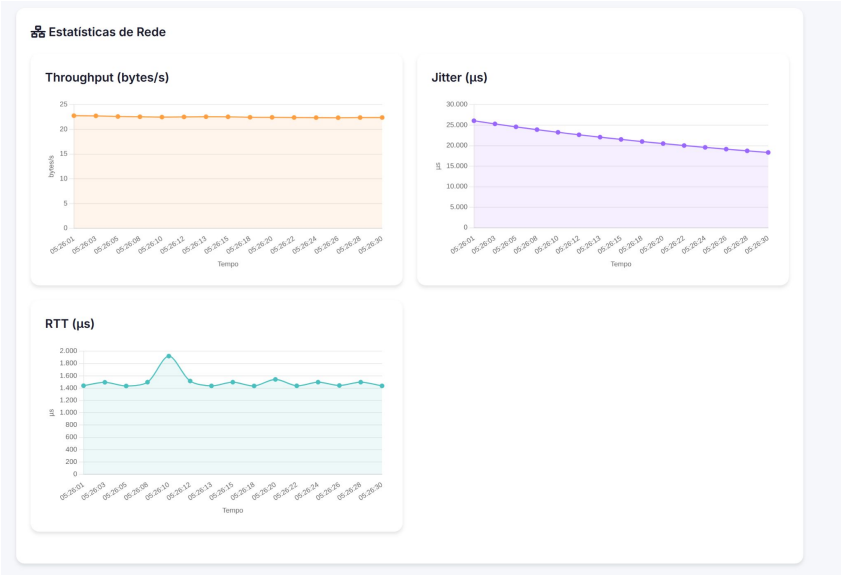
Pré-requisitos - Backend

- Java 25
- PostgreSQL se quiser persistência
- PostgreSQL se quiser persistência,
- Gradle,
- ESP32
  - DHT11 - sensor de temperatura e umidade utilizado como exemplo.
  - qualquer outro módulo que se queira controlar e interagir.

Pré-requisitos - ESP32

- Java 25
- PostgreSQL se quiser persistência
- PostgreSQL se quiser persistência,
- Gradle,
- ESP32
  - DHT11 - sensor de temperatura e umidade utilizado como exemplo.
  - qualquer outro módulo que se queira controlar e interagir.

Performance



Contribuição

Contribua ao projeto adicionando

- novas threads para novos sensores,
- Frontend mais robusto
- Implementando testes simulados para ambos os lados (Back e ESP32)